Artificial Intelligence Project--RLE and MIT Computation Center
Symbol Manipulating Language--Memo 14--The Wang Algorithm for
the Propositional Calculus Programmed in LISP

by

John McCarthy

This memorandum describes a LISP program for deciding whether an expression in the propositional calculus is a tautology according to Wang's algorithm. Wang's algorithm is an excellent example of the kind of algorithm which is conveniently programmed in LISP, and the main purpose of this memorandum is to help would-be users of LISP see how to use it.

1. The Wang algorithm. We quote from pages 5 and 6 of Wang's paper.

"The propositional calculus (System P)

Since we are concerned with practical feasibility, it is preferable to use more logical connectives to begin with when we wish actually to apply the procedure to concrete cases. For this purpose we use the five usual logical constants $\sim$(not), $\&$(conjunction), $\vee$ (disjunction), $\supset$ (implication), $\equiv$(biconditional), with their usual interpretations.

A propositional letter P, Q, R, M or N, et cetera, is a formula (and an "atomic formula"). If ϕ, ψ are formulae, then $\sim\phi, \phi \& \psi, \phi \vee \psi, \phi \supset \psi, \phi \equiv \psi$ are formulae. If π, ρ are strings of formulae (each, in particular, might be an empty string or a single formula) and ϕ is a formula, then π, ϕ, ρ is a string and $\pi \rightarrow \rho$ is a sequent which, intuitively speaking, is true if and only if either some formula in the string π (the "antecedent") is false or some formula in the string ρ (the "consequent") is true, i.e., the conjunction of all formulae in the antecedent implies the disjunction of all formulae in the consequent.

There are eleven rules of derivation. An initial rule states that a sequent with only atomic formulae (proposition letters) is a theorem if and only if a same formula occurs on

both sides of the arrow. There are two rules for each of the five truth functions-one introducing it into the antecedent, one introducing it into the consequent. One need only reflect on the intuitive meaning of the truth functions and the arrow sign to be convinced that these rules are indeed correct. Later on, a proof will be given of their completeness, i.e., all intuitively valid sequents are provable, and of their consistency, i.e., all provable sequents are intuitively valid.

P1. Initial rule: if λ, ζ are strings of atomic formulae, then $\lambda \longrightarrow \zeta$ is a theorem if some atomic formula occurs on both sides of the arrow.

In the ten rules listed below, λ and ζ are always strings (possibly empty) of atomic formulae. As a proof procedure in the usual sense, each proof begins with a finite set of cases of P1 and continues with successive consequences obtained by the other rules. As will be explained below, a proof looks like a tree structure growing in the wrong direction. We shall, however, be chiefly interested in doing the step backwards, thereby incorporating the process of searching for a proof.

The rules are so designed that given any sequent, we can find the first logical connective, i.e., the leftmost symbol in the whole sequent that is a connective, and apply the appropriate rule to eliminate it, thereby resulting in one or two premises which, taken together, are equivalent to the conclusion. This process can be repeated until we reach a finite set of sequents with atomic formulae only. Each connective-free sequent can then be tested for being a theorem or not, by the initial rule. If all of them are theorems, then the original sequent is a theorem and we obtain a proof; otherwise we get a counterexample and a disproof. Some simple samples will make this clear.

For example, given any theorem of "Principia", we can automatically prefix an arrow to it and apply the rules to look for a proof. When the main connective is , it is simpler, though not necessary, to replace the main connective by an arrow and proceed. For example:

*2.45. $\vdash : \sim(P \vee Q) \cdot \supset \cdot \sim P$,

*5.21: $\vdash : \sim P \,\&\, \sim Q \cdot \supset \cdot P \equiv Q$

can be rewritten and proved as follows.

*2.45 $\sim(P \vee Q) \rightarrow \sim P$ (1)

(1) $\rightarrow \sim P, P \vee Q$ (2)

(2) $P \rightarrow P \vee Q$ (3)

(3) $P \rightarrow P, Q$

VALID

QED

*5.21. $\rightarrow \sim P \,\&\, \sim Q \cdot \supset \cdot P \equiv Q$ (1)

(1) $\sim P \,\&\, \sim Q \rightarrow P \equiv Q$ (2)

(2) $\sim P, \sim Q \rightarrow P \equiv Q$ (3)

(3) $\sim Q \rightarrow P \equiv Q, P$ (4)

(4) $\rightarrow P \equiv Q, P, Q$ (5)

(5) $P \rightarrow Q, P, Q$

VALID

(5) $Q \rightarrow P, P, Q$

VALID

P2a. Rule $\rightarrow\sim$: If $\phi, \zeta \rightarrow \lambda, \rho$, then $\zeta \rightarrow \lambda, \sim\phi, \rho$.

P2b. Rule $\sim\rightarrow$: If $\lambda, \rho \rightarrow \pi, \phi$, then $\lambda, \sim\phi, \rho \rightarrow \pi$.

P3a. Rule $\rightarrow \&$: If $\zeta \rightarrow \lambda, \phi, \rho$ and $\zeta \rightarrow \lambda, \psi, \rho$, then $\zeta \rightarrow \lambda, \phi \,\&\, \psi, \rho$.

P3b. Rule $\&\rightarrow$: If $\lambda, \phi, \psi, \rho \rightarrow \pi$, then $\lambda, \phi \,\&\, \psi, \rho \rightarrow \pi$.

P4a. Rule $\rightarrow \vee$: If $\zeta \rightarrow \lambda, \phi, \psi, \rho$, then $\zeta \rightarrow \lambda, \phi \vee \psi, \rho$.

P4b. Rule $\vee\rightarrow$: If $\lambda, \phi, \rho \rightarrow \pi$ and $\lambda, \psi, \rho \rightarrow \pi$, then $\lambda, \phi \vee \psi, \rho \rightarrow \pi$.

P5a. Rule $\rightarrow \supset$: If $\zeta, \phi \rightarrow \lambda, \psi, \rho$, then $\zeta \rightarrow \lambda, \phi \supset \psi, \rho$.

P5b. Rule $\supset\rightarrow$: If $\lambda, \psi, \rho \rightarrow \pi$ and $\lambda, \rho \rightarrow \pi, \phi$, then $\lambda, \phi \supset \psi, \rho \rightarrow \pi$.

P6a. Rule $\rightarrow \equiv$: If $\phi, \zeta \rightarrow \lambda, \psi, \rho$ and $\psi, \zeta \rightarrow \lambda, \phi, \rho$, then $\zeta \rightarrow \lambda, \phi \equiv \psi, \rho$.

P6b. Rule $\equiv\rightarrow$: If $\phi, \psi, \lambda, \rho \rightarrow \pi$ and $\lambda, \rho \rightarrow \pi, \phi, \psi$, then $\lambda, \phi \equiv \psi, \rho \rightarrow \pi$. »

2. The LISP program. We define a function theorem [s] whose value is truth or falsity according to whether the sequent s is theorem.

The sequent

s: $\varphi_1,\ldots,\varphi_n \rightarrow \psi_1,\ldots,\psi_m$

is represented by the S-expression

s^1: (ARROW,($\varphi_1^1,\ldots,\varphi_n^1$),($\psi_1^1,\ldots,\psi_m^1$))

where in each case the ellipsis ... denotes missing terms.

Propositional formulae are represented as follows:

1. For "atomic formulae" (Wang's terminology) we use "atomic symbols" (LISP terminology).

2. The following table gives our "Cambridge Polish" way of representing propositional formulae with given main connectives.

1. $\sim\varphi$	becomes	(NOT, φ^1)
2. $\varphi \& \psi$	becomes	(AND, φ^1, ψ^1)
3. $\varphi \vee \psi$	becomes	(OR, φ^1, ψ^1)
4. $\varphi \supset \psi$	becomes	(IMPLIES, φ^1, ψ^1)
5. $\varphi \equiv \psi$	becomes	(EQUIV, φ^1, ψ^1)

Thus the sequent

$\sim P \& \sim Q \rightarrow P \equiv Q, R \vee S$

is represented by

(ARROW,((AND,(NOT,P),(NOT,Q)),((EQUIV,P,Q),(OR,R,S)))

The S-function theorem [s] is given in terms of auxiliary functions as follows:

```
theorem[s] = th1[NIL;NIL;cadr[s];caddr[s]]
th1[a1;a2;a;c] = [null[a]→th2[a1;a2;NIL;NIL;c];T→
member[car[a];c]∨[atom[car[a]→
th1[[member[car[a];a1]→a1;T→cons[car[a];a1]];a2
cdr[a];c];T→th1[a1;[member[car[a];a2]→a2;
T→cons[car[a];a2]];cdr[a];c]]
```

```
th2[a1;a2;c1;c2;c] = [null[c] → th[a1;a2;c1;c2];
atom[car[c]] → th2[a1;a2;[member[car[c];c1] → c1;T →
cons[car[c];c1]];c2;cdr[c]];T → th2[a1;a2;c1;[
member[car[c];c2] → c2;T →
cons[car[c];c2]];cdr[c]]]
th[a1;a2;c1;c2] = [null[a2] → ~null[c2] ∧ thr[
    car[c2];a1;a2;c1;cdr[c2]];T → thl[car[a2];a1;
    cdr[a2];c1;c2]]
```

th is the main predicate through which all the recursions take place. theorem, thl and th2 break up and sort the information in the sequent for the benefit of th. The four arguments of th are:

a1: atomic formulae on left side of arrow
a2: other formulae on left side of arrow
c1: atomic formulae on right side of arrow
c2: other formulae on right side of arrow

The atomic formulae are kept separate from the others in order to make faster the detection of the occurrence of formula on both sides of the arrow and the finding of the next formula to reduce. Each use of th represents one reduction according to one of the 10 rules. The formula to be reduced is chosen from the left side of the arrow if possible. According to whether the formula to be reduced is on the left or right we use thℓ or thr. We have

```
thℓ[u;a1;a2;c1;c2] =[
    car[u] = NOT → thlr[cadr[u];a1;a2;c1;c2]
    car[u] = AND → th2ℓ[cdr[u];a1;a2;c1;c2]
    car[u] = OR → th1ℓ[cadr[u];a1;a2;c1;c2] ∧ th1ℓ[
        caddr[u];a1;a2;c1;c2]
    car[u] = IMPLIES → th1ℓ[caddr[u];a1;a2;c1;c2] ∧ th1r[
        cadr[u];a1;a2;c1;c2]
    car[u] = EQUIV → th2ℓ[cdr[u];a1;a2;c1;c2] ∧ th2r[
        cdr[u];a1;a2;c1;c2]

]
```

```
thr[u;a1;a2;c1;c2] = [
    car[u] = NOT → th1ℓ[cadr[u];a1;a2;c1;c2]
    car[u] = AND → th1r[cadr[u];a1;a2;c1;c2] ∧ th1r[
                caddr[u];a1;a2;c1;c2]
    car[u] = OR → th2r[cdr[u];a1;a2;c1;c2]
    car[u] = IMPLIES → th11[cadr[u];caddr[u];a1;a2;c1;c2]
    car[u] = EQUIV → th11[cadr[u];caddr[u];a1;a2;c1;c2] ∧ th11[
                caddr[u];cadr[u];a1;a2;c1;c2]
```

The functions th1ℓ, th1r, th2ℓ, th2r, th11 distribute the parts of the reduced formula to the appropriate places in the reduced sequent.

These functions are

```
th1ℓ[v;a1;a2;c1;c2] = [atom[v]→member[v;c1] ∨
    th[cons[v;a1];a2;c1;c2];T → member[v;c2] ∨
    th[a1;cons[v;a2];c1;c2]

th1r[v;a1;a2;c1;c2] = [atom[v]→member[v;a1] ∨
    th[a1;a2;cons[v;c1];c2];T → member[v;a2] ∨
    th[a1;a2;c1;cons[v;c2]]

th2ℓ[v;a1;a2;c1;c2] = [atom[car[v]]→member[car[v];c1] ∨
    th1ℓ[cadr[v];cons[car[v];a1];a2;c1;c2];T → member[
        car[v];c2] ∨
    th1ℓ[cadr[v];a1;cons[car[v];a2];c1;c2]]

th2r[v;a1;a2;c1;c2] = [atom[car[v]]→member[car[v];a1] ∨
    th1r[cadr[v];a1;a2;cons[car[v];c1];c2];T → member[
        car[v];a2] ∨
    thr[cadr[v];a1;a2;c1;cons[car[v];c2]]]
```

$$th11[v1;v2;a1;a2;c1;c2] = [atom[v1] \rightarrow member[v1;c1] \vee$$
$$th1r[v2;cons[v1;a1];a2;c1;c2];T \rightarrow member[v1;c2] \vee$$
$$th1r[v2;a1;cons[v1;a2];c1;c2]]$$

Finally the function member is defined by

$$member[x;u] = \sim null[u] \wedge [equal[x;car[u]] \vee member[x;cdr[u]]]$$

3. The LISP Program as Written in S-expressions. The present section is redundant for those who fully understand LISP. In it we give the translation of the functions of the preceding section into S-expressions.

We have

```
DEFINE ((
(THEOREM (LAMBDA (S) (TH1 NIL NIL (CADR S) (CADDR S))))

(TH (LAMBDA (A1 A2 C1 C2) (COND ((NULL A2) (AND (NOT (NULL C2))
(THR (CAR C2) A1 A2 C1 (CDR C2)))) (T (THL (CAR A2) A1 (CDR A2)
C1 C2)))))

(TH1 (LAMBDA (A1 A2 A C) (COND ((NULL A)
(TH2 A1 A2 NIL NIL C)) (T
(OR (MEMBOB (CAR A) C) (COND ((ATOM (CAR A))
(TH1 (COND ((MEMBOB (CAR A) A1  ) A1)
(T (CONS (CAR A) A1))) A2 (CDR A) C))
(T (TH1 A1 (COND ((MEMBOB (CAR A) A2) A2)
(T (CONS (CAR A) A2))) (CDR A) C)))))))))

(TH2 (LAMBDA (A1 A2 C1 C2 C) (COND
((NULL C) (TH A1 A2 C1 C2))
((ATOM (CAR C)) (TH2 A1 A2 (COND
((MEMBOB (CAR C) C1) C1) (T
(CONS (CAR C) C1))) C2 (CDR C)))
(T (TH2 A1 A2 C1 (COND ((MEMBOB
(CAR C) C2) C2) (T (CONS (CAR C) C2)))
(CDR C)))))))
```

```
(THL (LAMBDA (U A1 A2 C1 C2) ( COND
((EQ (CAR U) (QUOTE NOT)) (TH1R (CADR U) A1 A2 C1 C2))
((EQ (CAR U) (QUOTE AND)) (TH2L (CDR U) A1 A2 C1 C2))
((EQ (CAR U) (QUOTE OR)) (AND (TH1L (CADR U) A1 A2 C1 C2)
(TH1L (CADDR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE IMPLIES)) (AND (TH1L (CADDR U) A1 A2 C1
C2) (TH1R (CADR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE EQUIV)) (AND (TH2L (CDR U) A1 A2 C1 C2)
(TH2R (CDR U) A1 A2 C1 C2) ))
(T (ERROR (LIST (QUOTE THL) U A1 A2 C1 C2)))
  )))

(THR (LAMBDA (U A1 A2 C1 C2) (COND
((EQ (CAR U) (QUOTE NOT)) (TH1L (CADR U) A1 A2 C1 C2))
((EQ (CAR U) (QUOTE AND)) (AND (TH1R (CADR U) A1 A2 C1 C2)
(TH1R (CADDR U) A1 A2 C1 C2) ))
((EQ (CAR U) (QUOTE OR)) (TH2R (CDR U) A1 A2 C1 C2))
((EQ (CAR U) (QUOTE IMPLIES)) (TH11 (CADR U) (CADDR U)
A1 A2 C1 C2))
((EQ (CAR U) (QUOTE EQUIV)) (AND (TH11 (CADR U) (CADDR U)
A1 A2 C1 C2) (TH11 (CADDR U) (CADR U) A1 A2 C1 C2) ))
(T (ERROR (LIST (QUOTE THR) U A1 A2 C1 C2)))
  )))

(TH1L (LAMBDA (V A1 A2 C1 C2  ) (COND
((ATOM V) (OR (MEMBER V C1)
(TH (CONS V A1) A2 C1 C2) ))
(T (OR (MEMBER V C2) (TH A1 (CONS V A2) C1 C2) ))
  )))

(TH1R (LAMBDA (V A1 A2 C1 C2) (COND
((ATOM V) (OR (MEMBER V A1)
(TH A1 A2 (CONS V C1) C2) ))
(T (OR (MEMBER V A2) (TH A1 A2 C1 (CONS V C2))))
  )))
```

```
(TH2L (LAMBDA (V A1 A2 C1 C2) (COND
((ATOM (CAR V)) (OR (MEMBER (CAR V) C1)
(TH1L (CADR V) (CONS (CAR V) A1 ) A2 C1 C2)))
(T (OR (MEMBER (CAR V) C2) (TH1L (CADR V) A1 (CONS (CAR V)
  A2) C1 C2)))
   )))

(TH2R (LAMBDA (V A1 A2 C1 C2) (COND
((ATOM (CAR V)) (OR (MEMBER (CAR V) A1)
(TH1R (CADR V) A1 A2 (CONS (CAR V) C1) C2 )))
(T (OR (MEMBER (CAR V) A2) (TH1R (CADR V) A1 A2 C1
(CONS (CAR V) C2))))
   )))

(TH11 (LAMBDA (V1 V2 A1 A2 C1 C2) (COND
((ATOM V1) (OR (MEMBER V1 C1) (TH1R V2 (CONS V1 A1) A2 C1
 C2)))
(T (OR (MEMBER V1 C2) (TH1R V2 A1 (CONS V1 A2) C1 C2)))
  )))

(MEMBER MEMBOB)
))  ()
```

This causes the functions mentioned to be defined. In our test run we next gave

```
TRACLIS ((TH)) ()
```

which caused the arguments and values of the function th to be printed each time the function came up in the recursion. Accidentally, it turns out that these arguments essentially constitute a proof in Wang's style of the theorem.

In order to apply the method to the sequent

$$p \rightarrow p \vee q$$

we write

```
THEOREM
((ARROW,(P),((OR,P,Q))))
()
```

4. Wang's Algorithm and Ambiguous Functions. This section contains a remark on the theory of computation suggested by the experience of programming Wang's algorithm. The functions given in section 2 owe some of their complexity to an effort to gain efficiency and could be made still more efficient at the cost of greater complexity. Namely, in the verbal description of the algorithm in section 1 it is not specified how the term in the sequent to be acted on first is chosen. This choice does not affect the ultimate result, i.e. whether a sequent is found to be a theorem, but does affect the time required.

This suggests that for the purposes of proving theorems about computable functions we introduce the concept of ambiguous function and certain of its properties.

An ambiguous function f assigns to certain n-tuplets $x_1,\ldots,x_n$ a value $f(x_1,\ldots,x_n)$. However, this value is not completely determined by the function f and the n-tuplet $x_1,\ldots x_n$ but may be any member of a set $R(\#,x_1,\ldots,x_n)$. No mechanism, probabilistic or otherwise is provided for deciding which element will be chosen. Ambiguous functions may be combined to get new functions by composition, conditional expressions, and recursion.

In order to consider the class $CA\{\tilde{f}\}$ of ambiguous functions computable in terms of a class of basic functions $\tilde{f}$, (This corresponds to $C\{\tilde{f}\}$ in [4]) we introduce the ambiguity operator amb. amb(x,y) is an ambiguous function of x and y whose value may be either x or y. Other ambiguities may be

introduced by using amb. For example we can define

$$\text{amblis}[\ell] = [\sim\text{null}[\ell]\wedge\text{null}[\text{cdr}[\ell]] \rightarrow \text{car}[\ell];$$
$$\sim\text{null}[\ell]\wedge\sim\text{null}[\text{cdr}[\ell]] \rightarrow \text{amb}[\text{car}[\ell];\text{amblis}[\text{cdr}[\ell]]]]$$

However, unless we have a list of all atomic symbols available we cannot define a function whose value is ambiguous among all atomic symbols.

We can introduce a transitive relation of descent between ambiguous functions. We say that f is a descendant of g, written

$$f \prec g$$

if for every x every possible value of $f(x)$ is also a possible value of $g(x)$. The operations used to define new functions in terms of old ones, i.e. composition, conditional expressions and recursion preserve descent in the following sense: If a function h_1 is defined by a recursive definition in terms of a function f_1 and other functions, and a function h_2 is defined in the very same way but with f_1 replaced by f_2; then from $f_1 \prec f_2$ it follows that $h_1 \prec h_2$.

A property P of functions such that if f satisfies P and $g \prec f$ implies that g satisfies P is called hereditary.

We apply these concepts to the example of Wang's algorithm as follows:

1. We define a predicate reduced[s] which is true if and only if the sequent has only atomic terms.

2. We define dup[s] which is true if the reduced sequent s has a common term on the right and left.

3. We define reduce1[s;term] and reduce2[s;term] which gives the two sequents into which the sequent s reduces when it is attacked at term (one of these may be trivial).

4. We define terms[s], the list of terms of the sequent.

Wang's algorithm for determining whether a sequent is a theorem

is given by

```
wang[s] = [reduced[s] → dup[s];T →
wang[reduced1[s;amblis[terms[s]]]]∧
wang[reduced2[s;amblis[terms[s]]]]]
```

This is an apparently ambiguous function because of the presence of amblis. However, it is not hard to show that actually it converges for each sequent s and that the result is T or F according to whether s is a theorem. This property is hereditary and hence applies to any descendant of the function wang.

Particular functions realizing Wang's algorithm choose the term to be reduced in a sequent in a sophisticated way. However, we need only show that they are descendants of wang in order to show that they give the right answer.

All this suggests that the concept of ambiguous function may have some importance in the theory of computation.

REFERENCES

1. Wang, Hao, "Toward Mechanical Mathematics", IBM Journal of Research and Development, Vol.4, No.1, January 1960.
2. McCarthy, John, "Recursive Functions of Symbolic Expressions and their Computation by Machine", Quarterly Progress Report No.53, Research Laboratory of Electronics, MIT, April 1959.
3. McCarthy, John, Fox, Phyllis, et. al.,"LISP Programmers' Manual", Artificial Intelligence Project, MIT, January 1960.
4. McCarthy, John, "Notes on the Theory of Computation", MIT Computation Center, to appear.

CS-TR Scanning Project
Document Control Form

Date : 11/30/95

Report # AIM-14

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)

☐ Other:_____

Document Information

Number of pages: 13(17-IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:

☒ Single-sided or

☐ Double-sided

Intended to be printed as :

☐ Single-sided or

☒ Double-sided

Print type:

☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MIMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages(by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-13)	1-13
(14-17) SCANCONTROL, TRGT'S (3)	

Scanning Agent Signoff:

Date Received: 11/30/95 Date Scanned: 12/13/95 Date Returned: 12/14/95

Scanning Agent Signature: Michael W. Cook

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**